Express Mail Label No. EL818624666US

HARDWARE IMPLEMENTATION OF AN APPLICATION-LEVEL WATCHDOG TIMER

CROSS-REFERENCE TO RELATED APPLICATIONS

[0001] Not applicable.

STATEMENT REGARDING FEDERALLY SPONSORED RESEARCH OR DEVELOPMENT

[0002] Not applicable.

BACKGROUND OF THE INVENTION

Field of the Invention

[0003] The present invention generally relates to watchdog timers for personal computer systems. More specifically, the preferred embodiment relates to the use of an application watchdog timer to monitor the uptime of individual applications running on a computer system.

Background of the Invention

[0004] Watchdog circuits are rather common in modern computer systems. A watchdog circuit is one way of creating a stable computing platform. In fact, when one speaks of a stable, robust computer system, the watchdog circuit is indirectly one of the reasons that the system has these attributes. Computer designers rely on the watchdog circuit to reset the system in the unfortunate event something goes wrong. If a computer system hangs or locks up, the watchdog circuit can perform a number of tasks, including logging error information, checking memory, and rebooting the system so the computer will be up and running again in a short amount of time.

[0005] A watchdog circuit typically is a timing circuit that measures a certain system activity or activities. If the system activity does not occur within a prescribed timer period, the watchdog circuit generates an output signal indicating that the activity has not occurred. In its simplest form, the watchdog timer insures that the system is operational. Modern watchdog circuits are capable of performing a variety of tasks, but the heart of a watchdog timer is essentially just a counter. The timer continually counts up or down using the system clock towards a predetermined value until one of two things happen. First, the counter can be cleared so that the amount of time required to count to the predetermined value is pushed back to the maximum value. For example, if a timer counts from a maximum value of 300 seconds towards a minimum value of zero seconds, then when the timer is cleared, the clock will revert back to the maximum value and continue counting down from 300 seconds. The clear command (sometimes referred to as "hitting the watchdog") is typically issued by the operating system ("OS"). Programmers will insert commands in the OS code instructing the OS to periodically hit the watchdog. Thus, as long as the OS is operating as intended, the watchdog timer will be cleared periodically and the timer never reaches the predetermined value.

[0006] The second thing that may happen as the watchdog timer is running is that the counter actually does reach the predetermined value. This obviously occurs if the watchdog is never hit and the timer is never cleared. In this case, the watchdog timer will issue a reset command to the system and the computer will reboot. This type of automatic recovery is particularly helpful in unmanned computer systems. Obviously, if a user is working at a computer system and the OS becomes unresponsive, the user can initiate the reset procedure themselves. If, on the other hand, the computer is generally unmanned and working as a server in a computer network, it may not be readily obvious that the computer has ceased normal operations. The first person affected by such

a condition will likely be a network user who discovers that they can't access a network database or perhaps their email. Thus, if a server becomes inoperative, the watchdog timer guarantees that the system will be up and running again in a short amount of time.

[0007] In their present configuration, conventional watchdog timers are certainly useful for their intended purpose. However, there are a number of drawbacks that can be improved upon by a more modern approach. From the perspective of server customers, the health of the OS is not necessarily the most important aspect of a network server. More often than not, a server actually exists to run a specific application and the proper operation of that application is the most important goal for the customer. Thus, if the key application or applications cease operation, but the OS effectively continues, the system will never reset and the customer experiences unwanted downtime.

[0008] Software solutions to the problem of monitoring applications have been proposed, but these implementations often require the existence of a separate watchdog application or service. Furthermore, these existing methods for monitoring applications are not robust as they require the watchdog application and the operating system to be operating correctly. A more efficient solution to this problem is to provide a hardware watchdog timer that is dedicated to the applications. This hardware is separate from the system watchdog timer and is capable of resetting the system in the event a key application becomes unresponsive. Likewise, if the OS is unresponsive, the system watchdog timer will also recover the application by forcing a system reset. In either case, the application and OS are fully monitored and system uptime is maximized.

[0009] It is desirable therefore, to develop an application-level watchdog timer that is capable of monitoring key applications and resetting the computer system in the event the applications become unresponsive. The application-level watchdog timer may work in conjunction with a

system level watchdog timer to provide a staggered level of protection that may advantageously improve computer server uptime.

BRIEF SUMMARY OF THE INVENTION

[0010] The problems noted above are solved in large part by an application watchdog, comprising a dedicated watchdog counter located in the hardware layer of a computer system and a watchdog driver operating in the kernel mode layer of the computer operating system. The watchdog driver comprises a system thread configured to monitor a plurality of designated user applications operating in the user mode of the computer operating system and a communication interface for transmitting a timer reset command to the dedicated watchdog counter. The watchdog driver uses a message passing interface for receiving periodic signals from each of the user applications.

[0011] If the system thread receives a message from each of the designated user applications within an allotted period of time, the watchdog driver sends a timer reset command to the dedicated watchdog counter. If the system thread does not receive a message from each of the designated user applications within the allotted period of time, the watchdog driver does not send a timer reset command to the dedicated watchdog counter. If the watchdog counter receives a timer reset command from the watchdog driver, the counter is reset to begin counting down from the maximum allotted period of time. However, if the watchdog counter does not receive the timer reset command from the watchdog driver, the counter is configured to restart the computer system when the counter expires.

[0012] The watchdog counter further comprises a timer value register that stores a digital representation of the maximum allotted period of time and a control and status register that comprises several different bit fields: a bit for enabling the application watchdog, a bit for counter

reset, bit fields for enabling early expiration warnings, and bit fields for early expiration warning signals. If the early expiration warnings are enabled, the counter is configured to transmit early expiration warnings to the rest of the computer system before the counter expires. These early warning messages may be maskable, non-maskable or system management interrupts sent to notify the system management software or firmware and are preferably delivered 9 seconds prior to system reset.

[0013] The application watchdog operates in conjunction with a conventional system watchdog that is configured to monitor the computer operating system for periodic activity. Both the application watchdog and the system watchdog are configured to reset the computer system such that if either watchdog does not receive a timer reset command within an allotted period of time, that watchdog may issue a system reset command. Alternatively, the watchdog devices may initiate a restart of the operating system or of individual applications. The watchdog devices may operate independent of one another with each device being selectably enabled and each capable of issuing a reset command.

[0014] Initialization of the watchdog driver comprises loading the watchdog driver as the operating system loads following a computer system boot and loading and creating an initial input/output control signal interface that establishes the message passing interface between the designated applications and the watchdog driver. The computer applications then initialize and register with the watchdog service. This process involves linking the application with a dynamic link library and calling the watchdog driver via the dynamic link library and through the initial input/output control signal interface to validate the message passing interface. The application preferably sends address and identification information to the watchdog driver. Lastly, the watchdog timer device is initialized by setting the timer initialization value in the timer value

register and setting the counter enable bit and early warning enable bits in the control/status register.

BRIEF DESCRIPTION OF THE DRAWINGS

[0015] For a detailed description of the preferred embodiments of the invention, reference will now be made to the accompanying drawings in which:

[0016] Figure 1 shows a simple computer network comprising a computer system in which the preferred embodiment may be implemented;

[0017] Figure 2 shows a block diagram of a computer system in which the preferred embodiment may be implemented;

[0018] Figure 3 shows a simplified ASM unit on which the preferred embodiment may be implemented;

[0019] Figure 4 provides a block diagram showing the implementation of the preferred embodiment with a conventional system watchdog timer;

[0020] Figure 5 shows a schematic displaying the hardware and software layer architecture of the preferred embodiment;

[0021] Figure 6 shows a flow chart describing the initialization and operation of the preferred embodiment; and

[0022] Figure 7 shows a the contents of the timer and control/status registers used in the preferred embodiment.

NOTATION AND NOMENCLATURE

[0023] Certain terms are used throughout the following description and claims to refer to particular system components. As one skilled in the art will appreciate, computer companies may

refer to a component by different names. This document does not intend to distinguish between components that differ in name but not function. In the following discussion and in the claims, the terms "including" and "comprising" are used in an open-ended fashion, and thus should be interpreted to mean "including, but not limited to...". Also, the term "couple" or "couples" is intended to mean either an indirect or direct electrical connection. Thus, if a first device couples to a second device, that connection may be through a direct electrical connection, or through an indirect electrical connection via other devices and connections.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

[0024] Turning now to the figures, Figure 1 shows an example of a simple computer network 10 comprising a plurality of computers. At least one of the computers 20 operates as a central server providing data to the other node computers 100, 120, which are connected to the same network 10. The central server 20 is coupled to the first computer 100 and the second computer 120 by network connections 122. Various other network components such as hubs, switches, modems, and routers may be included in the network 10, but are not shown in Figure 1. It is envisioned server 20 incorporates the preferred embodiment of the invention. Computers 100, 120 may preferably be "client" computers and may also implement the preferred embodiment. Although a client/server configuration is shown, the computer network may also be an enterprise network, a peer network, a wide area network, a web network or any other suitable network configuration.

[0025] The central server 20 preferably includes at least one input device such as a keyboard 30 and at least one output device such as a monitor 40. Other I/O devices such as a mouse, printer, keyboard, and speakers are certainly permissible and are perhaps desirable peripheral components.

[0026] Users working on computers 100, 120 may remotely access data such as file databases or software applications located on the server 20. Alternatively, software applications may be loaded

and run directly on the computers 100, 120, but licenses for the authorized use thereof are located on the central server 20. In either event, if a key application that is needed to provide data from the central server 20 to the network computers 100, 120 becomes unresponsive, that data will become unavailable and users on the network will be inconvenienced.

[0027] It can be appreciated therefore, that the ability to restart a server 20 if a key application becomes unresponsive provides certain advantages. The biggest advantage derives from the fact that an application failure may not result in an operating system failure. The preferred embodiment provides protection against this undesirable scenario and ensures that the network users are not inconvenienced for an unreasonably lengthy period of time.

[0028] Referring now to Fig. 2, a representative computer server system is illustrated. It is noted that many other representative configurations exist and that this embodiment is described for illustrative purposes. For the following discussion, the computer system of Fig. 2 is assumed to represent server computer 20, but one of skill in the art will recognize that the preferred embodiment may be implemented as part of any computer system. The computer system 20 of Fig. 2 preferably includes multiple CPUs 202 coupled to a bridge logic device 206 via a CPU bus 203. The bridge logic device 206 is sometimes referred to as a "North bridge" for no other reason than it often is depicted at the upper end of a computer system drawing. The North bridge 206 also preferably comprises a memory controller to access and control a main memory array 204 via a memory bus 205. The North bridge 206 couples CPUs 202 and memory 204 to each other and to various peripheral devices in the system via one or more high-speed, narrow, source-synchronous expansion buses such as a Fast I/O bus and a Legacy I/O bus. The North bridge 206 can couple additional "high-speed narrow" bus links other than those shown in Figure 2 to attach other bridge devices and other buses such as a PCI-X bus segment to which additional peripherals such as a

1Gigabit Ethernet adapter may be coupled. The embodiment shown in Figure 2 is not intended to limit the scope of possible server architectures.

[0029] The Fast I/O bus shown in Figure 2 may be coupled to the North bridge 206. In this preferred embodiment, the Fast I/O bus attaches an I/O bridge 214 that provides access to a high-speed 66Mhz, 64-bit PCI bus segment. A SCSI controller 215 preferably resides on this high speed PCI bus and controls multiple fixed disk drives 222. The high speed PCI bus also provides expansion slots 216 that permit coupling of peripheral devices that comply with the high speed PCI bus.

[0030] The Legacy I/O bus is preferably used to connect legacy peripherals and a primary PCI bus via a separate bridge logic device 212. This bridge logic 212 is sometimes referred to as a "South bridge" reflecting its location vis-à-vis the North bridge 206 in a typical computer system drawing. An example of such bridge logic is described in U.S. Patent No. 5,634,073, assigned to Compaq Computer Corporation. The South bridge 212 provides access to the system ROM 213 and provides a low-pin count ("LPC") bus to legacy peripherals coupled to an I/O controller 226. The I/O controller 226 typically interfaces to basic input/output devices such as a floppy disk drive 228, a keyboard 30, a mouse 232 and, if desired, various other input switches such as a power switch and a suspend switch (not shown). The South bridge 212 also may provide one or more expansion buses, but preferably provides a 32-bit 33Mhz PCI bus segment on which various devices are disposed. It should be noted that the Legacy I/O bus may be narrower than other "high speed narrow" buses if it only needs to satisfy the bandwidth requirements of peripherals disposed on the 33Mhz, 32-bit PCI bus segment.

[0031] Various components that comply with the bus protocol of the 33Mhz, 32-bit PCI bus may reside on this bus, such as a video controller 208 and a network interface card ("NIC") 217. The

video controller 208 preferably drives a video display device 40 while NIC 217 is coupled to a network 218 for communication with other computers. These components may be integrated onto the motherboard as presumed by Fig 2, or they may be plugged into expansion slots 210 that are connected to the PCI bus. In addition to the NIC 217 and video controller 208, an Advanced Server Management ("ASM") unit 230 is also disposed on the 33Mhz, 32-bit PCI bus. The ASM unit 230 includes a system watchdog of the type that is found in many conventional computer systems. An example of such a watchdog is the Automatic Server Recovery ("ASR") watchdog found in some Compaq Computer Corporation servers. In the preferred embodiment, the application watchdog is also located on the ASM unit 230. A more detailed description of the ASM unit 230 is provided below in the discussion of Figure 3.

[0032] Figure 3 represents a simplified block diagram showing some of the various functions provided by the ASM unit 230 in the preferred embodiment. The ASM is a multipurpose management ASIC chip that provides various management facilities in addition to the watchdog device 330. In the preferred embodiment, the ASM ASIC includes an I/O CPU (or I/O processor) 320 that is used to provide intelligent control of the management architecture in the server 20. In addition to the CPU 320, the ASM 230 also preferably includes one or more out-of-band communication interfaces such as a Network Interface 300 and/or serial port device (not shown). These communication interfaces 300 permit out-of-band communication with the ASM 230 to enable remote monitoring, control, and detection of various system management events, including those generated by the watchdog device 330.

[0033] The ASM 230 also preferably includes an Integrated Remote Console ("IRC") 310. The IRC 310 provides the hardware facilities necessary to enable system management firmware, preferably executing on the CPU 320, to redirect console input (*e.g.*, keyboard 30 and mouse 232)

as well as console output 40 on the managed server 20 to a remote authorized user through one of the out-of-band communication interfaces 300 mentioned above.

[0034] The last function shown in Figure 3 is the Watchdog device 330, which incorporates a conventional system watchdog timer as well as the application watchdog timer in accordance with the preferred embodiment. The ASM unit 230 may also perform any number of additional tasks including system support functions and providing UART serial communication capabilities (not shown). In short, the ASM unit 230 is a design specific device that is fully configurable to a design engineer's requirements. The preferred embodiment of the application watchdog is just one of many functions that are executed by the ASM unit 230.

[0035] As mentioned above, the application watchdog timer supplements a conventional system watchdog timer. The interrelation of the two watchdog timers is shown in Figure 4. In Figure 4, the system watchdog 400 and the application watchdog 410, each operate as a conventional watchdog, counting down from some predetermined reset value until the watchdog is either cleared or until the timer reaches its final value, thus triggering a system reset command ("SYSRST#"). The system watchdog 400 responds to clear commands from the operating system whereas the application watchdog responds to clear commands from individual computer applications. The watchdog timer also monitors a PGOOD power supply signal, which indicates the computer power supply is operating as expected. If either watchdog timer 400, 410 is not cleared (by the operating system or by the applications) in the predetermined reset time or if the PGOOD signal is not valid, a system reset command SYSRST# is issued. Reset logic 420 receives and interprets the PGOOD and reset commands from the watchdog timers 400, 410 and delivers the SYSRST# command when appropriate. In addition to transmitting a SYSRST# command, the watchdog device 330

may also be configured to transmit maskable event notification interrupts to the I/O CPU 320 indicating which of the watchdog timers 400, 410 expired and thus initiated the reset procedure.

[0036] It should be noted that a reset command from either watchdog timer 400, 410 under normal operating conditions is sufficient to reset the system. Thus, the preferred embodiment provides protection against application failures as well as operating system failures. It should also be noted that the watchdog devices 400, 410 operate independent of one another and each may be selectably enabled or disabled as described below. In addition to a system reset as indicated by the SYSRST# signal shown in Figure 4, the watchdog device 330 may also initiate alternative reset procedures, such as an operating system reset or an individual application kill/reset procedure.

[0037] Figure 4 also shows early expiration signals that may be issued by the watchdog timers 400, 410. The watchdog timers 400, 410 are configurable to send these early warning signals before the respective timers expire. Warning logic 430 receives the early warning signals and delivers interrupts to the operating system and/or system management software as a warning that the watchdog timer is about to expire. Additionally, the watchdog 330 may also transmit warning interrupts to the I/O CPU 320. These interrupts allow the system to perform any necessary tasks, such as saving a memory context or system information prior to the upcoming system reset. The exact nature of these early expiration interrupts is discussed in more detail below.

[0038] Referring now to Figure 5, a schematic showing the system architecture of the preferred embodiment is shown. The preferred embodiment is described for, but not limited to, a Windows NT environment. The three main levels shown in Figure 5 represent the hardware/software protection layers in a conventional computer system running the Windows NT operating system. The NT environment provides two software protection levels: Ring 0 and Ring 3. Other systems may provide up to 4 or more protection levels. The Ring 0 protection level, sometimes called the

kernel mode or supervisor mode, is the most highly protected ring in which an application or service can run. The Ring 3 protection level, sometimes called the application level or user mode, is the least protected ring. Applications running in Ring 3 cannot physically access memory space in the more highly protected Ring 0 layer. Any communication between applications running in Ring 3 and services in Ring 0 must use a message passing service. This design prevents user applications from interfering with the core NT operating system.

[0039] Also shown in Figure 5 is a Hardware layer, which represents the physical computer system hardware such as the CPUs, timer devices, and watchdog devices. For the purposes of illustrating the preferred embodiment, Figure 5 shows only the application watchdog timer device 410. Also included in Figure 5 is a Hardware Abstraction Layer ("HAL") 510, which is used to prevent hardware dependence and provide an isolation layer between the hardware and software. The HAL 510 operates at the Ring 0 level and translates low-level operating system functions into instructions understandable by the physical system hardware.

[0040] Another aspect of Figure 5 that is common to conventional NT system architectures is the location and execution of user applications 520, 530 in the Ring 3 protection layer. As discussed above, the protection levels are set up to ensure a stable operating system environment. In order to provide access to OS functions and data structures, a set of dynamic link libraries ("DLL") 540 are linked as extensions to the applications. The DLLs 540 may be shared between applications 520, 530 or may be uniquely related to a particular application. The applications 520, 530 and DLL 540 are typically linked at application load time. Furthermore, a message passing interface 550 is used to permit communication between the applications 520, 530 in the application layer and kernel mode drivers in the Ring 0 layer. The message passing interface 550 may be implemented as

shared memory queues, which transmit communication signals as well as manage any asynchronous inter-layer timing differences.

[0041] The above described architecture will now be supplemented with a description of the unique aspects and advantages of the preferred embodiment. Among the required components for the application watchdog is a kernel mode driver 560 with a system thread 570. The system thread 570 processes information from and communicates with the message passing interface 550, which is situated between protection levels. The application watchdog driver 550 mirrors those drivers that already exist in systems that provide a system watchdog driver to monitor the operating system. However, in this preferred embodiment, the clear commands that reset the watchdog timer originate from user level applications 520, 530. These clear commands are interpreted by the system thread 570 in the watchdog driver 560, which then issues a command (via the HAL 510) to clear the timer device 410. Thus, the timer device 410 and watchdog driver 560 shown in Figure 5 are dedicated to the applications 520, 530.

[0042] Referring now to Figure 6, a simplified flow chart describing the initialization and operation of the preferred embodiment is shown. The following description includes references to the watchdog system architecture as shown in Figure 5. The START procedure 600 begins during a computer system reset. This reset may be a cold boot, warm boot, or perhaps even a system reset initiated by the system level or application level watchdog timers. After the computer completes the boot operation and executes the POST operation, the operating system will load and initialize 610. During OS initialization 610, the application watchdog driver 560 uses I/O control calls ("IOCTLs") to establish the appropriate message passing interface 550. Once the OS is initialized and running, the key user applications 520, 530 are started and initialized 620.

[0043] It is envisioned that the watchdog driver 560 need not monitor all applications, but it is certainly possible to do so. In the preferred embodiment, the key user applications 520, 530 will be designated by the user and only these applications will request watchdog support. Once a key application is linked to an appropriate DLL 540, the application will call into the DLL 540, which in turn, will make initialization IOCTL calls into the watchdog driver 560 to verify a connection through the message passing interface 550. Once this interface is established, no further IOCTL calls will be required. The initialization IOCTL calls will likely have pointers, process id's, and callback addresses associated with the user applications 520, 530. The watchdog driver 560 contains a list and monitors each of the key user applications 520, 530 and clears the watchdog timer 410 when periodic messages are received from all applications in this list.

[0044] In addition to the OS initialization 610 and application initialization 620, the application watchdog timer device must be initialized 630. This initialization is consists of setting appropriate bits in a timer value register and a control and status register (shown in Figure 7) within the watchdog device. The timer value register is a 16-bit counter that counts down to a system reset. The control and status register is an 8-bit configuration register that enables the application watchdog and the early expiration warning interrupts. The control and status register also includes a timer reset field. The timer value register is initialized by writing the initial count value. The control and status register is initialized by setting an enable bit and optionally setting an early warning enable bit. Additional information regarding the register contents is provided below.

[0045] During runtime operation the user application sends messages periodically through the message passing interface 550. The watchdog driver system thread 570 will asynchronously monitor the interface 550 for periodic messages from the applications 520, 530. If the watchdog driver 560 detects messages 640 from all applications 520, 530, the driver 560 issues the clear

command 642 to the watchdog timer 410 and continues monitoring the shared memory queues 550 for the periodic messages. If the watchdog driver 560 does not detect a message from either of the applications 520,530 for a predetermined period of time, the driver 560 withholds the timer clear signal. As the watchdog timer 410 reaches the 9 second early warning threshold, the watchdog driver 560 issues the appropriate early warning signals 644. If the watchdog counter expires, the driver 560 issues a reset command 650. In other words, the watchdog driver 560 must receive signals from all registered applications 520, 530 before the watchdog clear command is issued to the watchdog timer 410. This process continues until the application 520, 530 is manually closed down or the computer system or operating system is shut down 660. A graceful termination of the application 520, 530 will not induce any watchdog events because the application de-registers from the watchdog list monitored by the driver 560. In the event of an operating system shutdown, or computer system shutdown, the operating system issues commands to the application to shut down. In response, the application 520, 530 de-registers from the watchdog list. That is, the application 520, 530 directs the watchdog driver 560 to remove that program's registration entry so that the watchdog driver 560 no longer looks for periodic messages from that application 520, 530. If all applications 520, 530 terminate, the watchdog list becomes null and the watchdog timer 410 itself is preferably disabled.

[0046] It is envisioned that the periodic signals sent by the applications 520, 530 will be initiated by commands embedded in the computer application software. These commands will be directed at the shared memory queues 550 for the purpose of clearing the application watchdog timer. It is feasible however, that the commands be sent by instructions in the DLL 540 or as part of normal communication with other parts of the computer including the CPUs, system memory, or the OS. In this case, the watchdog driver system thread 570 acts as a passive observer checking for activity

from the applications 520, 530. Other embodiments in accordance with the above teachings are certainly feasible.

[0047] Referring now to Figure 7, the contents of the application watchdog timer value register 700 and control/status register 710 are shown. As mentioned above, the timer value register is a countdown register that decrements from an initial value to a final system reset value. The register is 16-bits wide and each bit represents 128 msec. Thus, the timer, once enabled, will decrement every 128 msec unless the timer is cleared. When this timer reaches zero, the reset signal is asserted. The 16-bit register yields a range of 128 msec to approximately 140 minutes. Writes to this register set the initialization start value for the timer. Reads of the register return the current timer value in 128 msec units.

[0048] The control/status register 710 is an 8-bit register and contains at least 6 used bit fields. As discussed above, the enable bit enables the timer countdown sequence. Setting this bit will automatically clear the timer to the value programmed in the timer value register. The reload bit is a timer clear bit. Writing a one to this location will reload the timer with its initialization value. This bit is self clearing. The NMIEN and SMIEN bits enable different early expiration warning interrupts. In the preferred embodiment, the NMIEN bit is used to enable the generation of warning NMI (non-mask interrupt) whenever the timer reaches 9 seconds from expiration. If enabled, the NMISTAT bit is used by system management software to detect that the application watchdog timer is about to expire. Similarly, the SMIEN bit is used to enable the generation of a warning SMI# (system management interrupt) signal when the timer reaches 9 seconds from expiration. If enabled, the SMISTAT bit is used by SMM (system management mode) firmware to detect that the timer is about to expire. Bit locations 4 and 5 are reserved for features not presently incorporated in the preferred embodiment, but may be used for other interrupt signals, including

maskable interrupts. In general, the early warning interrupt may be any suitable maskable, non-maskable or system management interrupt.

[0049] As mentioned above, the watchdog 330 may be additionally configured to transmit event notification interrupts to the I/O CPU 320 residing on the ASM ASIC 230. The I/O CPU 320, which operates independently of the main CPU 202 and operating system, may wish to monitor these system events for the purpose of logging or transmitting system management notification alerts. If desired, these event notification interrupts may be configured and initialized much like the NMI and SMI interrupts described above. For instance, a mask register may be used to enable early warning notification and system reset notification interrupts for each watchdog. Hence, for each watchdog (application and system), the mask register may include a bit to enable early warning notifications and a separate bit to enable system reset notifications. Similarly, an event status register comprising corresponding bits may be used to indicate if the early warning or reset time periods expire for either watchdog.

[0050] It should also be noted that the 9 second early expiration warning is set for practicality and convenience reasons. There is no reason why this period cannot be extended or shortened to other periods of time. Furthermore, this time period is preferably hard coded into the registers, but it is also envisioned that the expiration time may be altered via a user-interactive software menu.

[0051] The above discussion is meant to be illustrative of the principles and various embodiments of the present invention. Numerous variations and modifications will become apparent to those skilled in the art once the above disclosure is fully appreciated. For example, since the watchdog driver 560 is capable of monitoring several applications, the watchdog system may be configured to provide a user interface to establish priority among the applications. For instance, some sort of policy control may be added that allows the alarm timer events to be delayed

more for one application compared to others. This will provide some measure of certainty to ensure that an application has hung before it is restarted. It is intended that the following claims be interpreted to embrace all such variations and modifications.